Programming in C/C++
Exercises set five: grammatical parsers

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

February 23, 2017

Exercise 30, calculator

We got a calculator grammar.

Code listings

Listing 1: grammar.gr

```
1  %baseclass-preinclude   cmath
2  %stype double
3
4  %token NUMBER
5  %left NEGATION
6  %left '*'
7  %left '/'
8  %left '+'
9  %left '-'
10 %left '$'
11
12 %%
13
14 input:
15         // empty
16 |
17         input line
18 ;
```

```
19
20 line:
21         '\n'
22 |
23         expr '\n'
24         {
25             std::cout << "\t" << $1 << '\n';
26         }
27 |
28   error '\n'
29 ;
30
31 expr:
32         NUMBER
33 |
34         expr '+' expr
35         {
36             $$ = $1 + $3;
37         }
38 |
39         expr '-' expr
40         {
41             $$ = $1 - $3;
42         }
43 |
44         expr '*' expr
45         {
46             $$ = $1 * $3;
47         }
48 |
49         expr '/' expr
50         {
51             $$ = $1 / $3;
52         }
53 |
54         '-' expr %prec NEGATION
55         {
56             $$ = -$2;
57         }
58 |
```

```
59    '$' expr
60    {
61        $$ = sqrt($2);
62    }
63  |
64          '(' expr ')'
65          {
66              $$ = $2;
67          }
68  ;
```

Exercise 31, conflicts

We fixed numerous conflicts.

First grammar

The first revision of the grammar solves the reduce/reduce conflicts that are caused by the `NUMBER` branches in the `expr` and `number` nonterminals. To solve this we removed the `expr` case because it is already represented in `number`.

```
1  %token NUMBER
2
3  %%
4
5  expr:
6      number
7  |
8      expr '+' expr
9  |
10     expr '-' expr
11 |
12   // empty
13 ;
14
15 number:
16     NUMBER
17 ;
```

Second grammar

In this grammar we also solved the shift/reduce conflicts which were caused because '+' and '-' have equal priority and no explicitly specified resolution for these types of conflicts. We set them to equal priority but reduce and not shift as is customary with these operators.

```
1  %token NUMBER
2
3  %left '+' '-'
4
5  %%
6
7  expr:
8      number
9  |
10     expr '+' expr
11 |
12     expr '-' expr
13 |
14   // empty
15 ;
16
17 number:
18     NUMBER
19 ;
```

Exercise 32, priorities

We added a new priority to a grammar.

Code listings

Listing 2: grammar.gr

```
1  %token NR
2
3  %left NOTEQUAL
4  %left '+'
5  %left '*'
```

```
6  %left '^'
7  %right '-'
8
9  %%
10
11 expr:
12   NR
13 |
14   '-' expr
15 |
16   expr '+' expr
17 |
18   expr '*' expr
19 |
20   expr NOTEQUAL expr
21 |
22   expr '^' expr
23 ;
```

Exercise 35, separated lists

We fixed the list grammar so it does what we want it to.

Design

To accomodate multiple datatypes we use listtokens instead of WORDs. To allow empty lists, lists can exist out of nothing (empty OR). Because a list with one item can't be distinguished as a separated list we just count it as a normal list. A separated list is thus counted as `[listtoken ',' listtoken]` with optional repetition of `[listtoken ',']`. We abbreviated these nonterminals as `sepstart` and `sepend`. To simplify some more we defined `[listtoken ',']` as `septoken`.

Code listings

Listing 3: grammar.gr

```
1  %token WORD
2  %token INT
3  %token FLOAT
4
```

```
%%

list:
  plain
|
  separated
|
  // empty
;

plain:
  plain
  listtoken
|
  listtoken
;

separated:
  sepstart
  sepend
;

sepend:
  sepend
  septoken
|
  // empty
;

sepstart:
  listtoken
  septoken
;

septoken:
  ','
  listtoken
;

listtoken:
```

```
45    WORD
46  |
47    INT
48  |
49    FLOAT
50  ;
```